

**Promoción del desarrollo de SW libre en un entorno de
calidad y confianza adaptando las metodologías, procesos,
modelos de negocio y últimas tecnologías**

FIT-350503-2007-7

**Entregable D4.2.2
Guía para el desarrollo de gadgets válidos
para la plataforma**

Universidad Rey Juan Carlos
Yaco Sistemas

Fecha límite del entregable: 29/02/2008

Fecha de entrega: 14/02/2008

Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-sa/3.0/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Este trabajo está parcialmente financiado por el Ministerio de Industria, Turismo y Comercio español.

Microsoft, Windows y el logo de Windows son marcas registradas de Microsoft Corporation.

Macintosh, Apple y el logo de Apple son marcas registradas de Apple Computer, Inc.

Historial de Cambios

Versión	Fecha	Estado	Autor (Partner)	Descripción
0.1	20/11/2007	Borrador	Sergio García Bravo (URJC)	Edición de la parte relacionada con contenido estático
0.5	27/11/2007	Borrador	Javier de la Rosa (Yaco)	Recopilación del contenido enviado por los partners y edición de la parte relacionada con contenido dinámico
0.9	28/12/2007	Borrador	Javier de la Rosa (Yaco)	Documento finalizado
1.0	14/02/2008	Final	Rafael Fernández (UPM)	Primera revisión del documento

RESUMEN EJECUTIVO

Este informe recopila en un único documento dos de los métodos posibles para la correcta implementación de gadgets válidos para la plataforma. A modo de tutoriales, recorre las características a tener en cuenta para desarrollar tanto gadgets estáticos, incluyendo una extensa descripción de las librerías de apoyo a utilizar, como gadgets dinámicos, siendo la plataforma de *mashups* EzWeb la escogida para la muestra.

Información del Documento

Proyecto FIT Número	FIT-350503-2007-7	Acrónimo	Vulcano
Título completo	Promoción del desarrollo de SW libre en un entorno de calidad y confianza adaptando las metodologías, procesos, modelos de negocio y últimas tecnologías		
URL	http://www.ines.org.es/vulcano		
URL del documento			

Entregable	Número	D4.2.2	Título	Guía para el desarrollo de gadgets válidos para la plataforma
Paquete de Trabajo	Número	4	Título	Arquitectura E2.0 para la forja Vulcano
Tarea	Número	T4.2	Título	Integración en la parte cliente

Fecha de Entrega	Contractual	29/02/2008	Entregado	14/02/2008
Estado	Versión 1.0, 28/12/2007		final <input checked="" type="checkbox"/>	
Tipo	Informe			
Nivel de Diseminación	Público <input type="checkbox"/> Consorcio <input type="checkbox"/>			
Resumen (para diseminación)	Guía para el desarrollo de gadgets válidos para la plataforma, ya sea para un entorno estático como para la nueva plataforma EzWeb			
Palabras Clave	Desarrollo de gadgets, guía de desarrollo			

Autores (Partner)	Universidad Rey Juan Carlos, Yaco Sistemas			
Responsable de Autoría	Javier de la Rosa		Email	jrosa@yaco.es
	Partner	Yaco Sistemas	Tfno	+34 95 470 00

TABLA DE CONTENIDOS

<u>RESUMEN EJECUTIVO.....</u>	<u>3</u>
<u>TABLA DE CONTENIDOS.....</u>	<u>5</u>
<u>1 INTRODUCCIÓN.....</u>	<u>6</u>
<u>2 DESARROLLO DE PLATAFORMA ESTÁTICA.....</u>	<u>7</u>
<u>3 DESARROLLO DE PLATAFORMA DINÁMICA.....</u>	<u>19</u>
<u>4 REFERENCIAS</u>	<u>37</u>

1 INTRODUCCIÓN

El continuo aumento del desarrollo de proyectos de software libre, en los cuales participan empresas de forma cooperativa, ha incentivado la creación de herramientas que organicen el trabajo y proporcionen el material necesario para su correcto desarrollo. Para ello se han ideado gestores de proyectos como pueden ser Gforge¹, Sourceforge² o incluso Trac³ (aunque este último sea más bien una herramienta de ticketing).

Estas herramientas plantean un problema que se ha ido expandiendo con el paso del tiempo. Debido al gran uso que están recibiendo, los desarrolladores suelen tener albergados proyectos en varios gestores, con lo que la administración de todos ellos se puede volver, en algunos casos, costosa. Con el objetivo de solucionar este problema surge EzForge⁴: una plataforma capaz de englobar todos los proyectos de un desarrollador y permitir administrarlos desde una única interfaz. En un principio con soporte para Gforge y Trac, pero en el futuro se tiene previsto ampliar esta gama de gestores.

Someramente, la estructura de la nueva forja se compone de básicamente de tres partes. La primera de ellas está formada por adaptadores que extraen información de los servicios finales como pueden ser Gforge, Trac, etc. Estos datos son tratados por una segunda capa intermedia que los envía a una última capa que los mostrará por pantalla. Este último paso se puede realizar de diversas maneras, siendo una de ellas, y la que se propone en este documento, el uso de gadgets para ese cometido. Un gadget es una pequeña porción de código que puede ser instalada o ejecutada (normalmente dentro de una página web basada en HTML) directamente por el usuario final, sin necesidad de cualquier tipo de preparación previa. Los gadgets derivan de la idea existente desde hace años del código reutilizable, y representan la vista que se le ofrece al usuario del servicio, fuente de datos, u otros gadgets desde los que se obtiene la información. Para el desarrollo de los mismos se va a utilizar el lenguaje interpretado JavaScript. Como características principales se podrían destacar la ejecución en cliente, es decir, en los navegadores y que no requiere compilación. La sintaxis es semejante a lenguajes como pueden ser Java o C. Para facilitar aun más el desarrollo de gadgets a desarrolladores inexpertos en lenguaje JavaScript, se van a proporcionar a lo largo del documento algunas directivas para la implementación que estarán basadas en varios framework como son Ext⁵ y Prototype⁶.

El propósito del documento es acercar al lector al desarrollo de gadgets en distintos entornos, estático y dinámico. La diferencia conceptual entre ellos es simple, en un entorno dinámico se posibilita la inserción y modificación de gadgets por el usuario desde el navegador sin tener que variar el código de las aplicaciones. Para ello se va a contar con la plataforma de mashup EzWeb como contenedor de gadgets. Estas plataformas son aplicaciones web híbridadas, es decir, lugares web que usan contenido de más de una fuente para crear un nuevo servicio completo.

Las aplicaciones que se describen a continuación están carentes en muchos aspectos de lógica, y únicamente se dedican a mostrar los datos que le llegan en formato XML. La simplicidad de estos gadgets permite que la portabilidad de los mismos sea sencilla y en muchos casos inmediata. Por ejemplo un gadget desarrollado para un entorno estático, como puede ser un documento HTML, debería ser fácilmente portable a EzWeb.

¹<http://gforge.org/>

²<http://sourceforge.org/>

³<http://trac.edgewall.org/>

⁴<https://forge.morfeo-project.org/wiki/index.php/EzForge>

⁵<http://www.extjs.com>

⁶<http://prototypejs.com>

2 DESARROLLO DE PLATAFORMA ESTÁTICA

2.1 Criterios generales

El propósito que pretende la plataforma estática de desarrollo de gadgets es agilizar y facilitar al desarrollador la implementación de los mismos. Para ello, es necesario que tengan una interfaz simple y proporcionen únicamente la funcionalidad necesaria. Este cometido ha sido logrado gracias a las facilidades que aporta el framework Ext.

La plataforma estática consta de una página web dividida en varias capas, *divs*, en las cuales se alojarán los gadgets, que no serán más que pequeños códigos javascript. El desarrollo de éstos se basa en el correcto tratamiento de los XML devueltos por la plataforma EzForge, ya que únicamente tendrán la función de dar formato y mostrar elementos en pantalla.

2.2 Framework

Ext es un framework para la construcción de aplicaciones web que se sitúa en el lado del cliente. Inicialmente, se ideó como una pequeña extensión de utilidades para la librería YUI⁷ (Yahoo! User Interface), llamándose yui-Ext. Pero con el tiempo fue creciendo y ganando en popularidad, debido a lo cual cambió su nombre y se denominó Ext.

Este framework requiere librerías base para poder funcionar, es decir, no es totalmente independiente, sino que se apoya en otras librerías también escritas en Javascript. En estos momentos, la plataforma tiene implementados adaptadores para YUI, jQuery y Prototype/Script.aculo.us. El uso de estas librerías aporta al desarrollador una serie de facilidades que le permiten obviar muchos aspectos. Por ejemplo, gracias a Ext, podemos realizar peticiones asíncronas de datos simplemente invocando a una función. Además de esto, proporciona clases que facilitan el tratamiento de XML, que permiten realizar interfaces intuitivas para el usuario de manera rápida, etc.

En la web del framework se encuentran las especificaciones de su API con una gran cantidad de ejemplos, además de tutoriales, tanto generales como de los aspectos más requeridos de Ext. En estos momentos está en desarrollo la versión 2.0 del framework, por lo que se ha decidido utilizar la versión estable 1.1.1.

2.3 Implementación de gadgets

En este pequeño tutorial se supondrá que el desarrollador tiene unas nociones básicas sobre el lenguaje javascript, HTML y XML. El primer paso es la obtención de las librerías correspondientes al framework Ext de la página web oficial.

Inicialmente se va a realizar un repaso de las clases básicas necesarias para el desarrollo de gadgets, y después se mostrarán unos ejemplos explicativos. Las clases que se van a utilizar en este tutorial son un fichero HTML, "Tutorial.html", y un fichero JavaScript, "Tutorial.js", opcionalmente se puede utilizar CSS para albergar los estilos de nuestra aplicación. En el primero de ellos es necesario incluir las cabeceras que hacen referencia a Ext, por ejemplo:

⁷<http://developer.yahoo.com/yui/>

```

! "#$% &"( " ) *$ + , " - &
& . /'000 0 1'2' +' 3 3&4
4
34
. 5 6& 7. & 6& 8' 9 6" :;&4
4% ' 4
% 3 "% 3 8/ 4
. 7.6& 8' . & 6&8 ,,, ' 3 . '7 '7 &4' .4
. 7.6& 8' . & 6&8 ,,, ' 3 . '7 ' 8 7 3 . &4' .4
. 7.6& 8' . & 6&8 ,,, ' 8 3 1 &4' .4
. 7.6& 8' . & 6& &4' .4

% 3 8 7 / 4
6& 7 & 7.6& 8' & <6&8 ,,, ' ' 8 &4
6& 7 & 7.6& 8' & <6&8 ,,, ' '7 &4
' 34
374
' 374
' 4

```

En este caso se ha creado una carpeta “Ext-1.1.1” en la que se encuentran todos los archivos pertenecientes al framework. Los ficheros “Tutorial.js” y “Tutorial.html” se encuentran en el mismo nivel que la carpeta. Después de crear los links con los ficheros necesarios de la librería se puede comenzar a crear el fichero JavaScript.

El método por el cual inicia la ejecución Ext se denomina *onReady*. Éste es llamado automáticamente una vez que el DOM ha sido cargado por completo, garantizando que cualquier elemento de la página que se desee referenciar ya estará disponible cuando se ejecute el *script*. Para comprobar el correcto funcionamiento de la librería se aconseja crear un JavaScript similar al siguiente:

```

8 2 37=< => ?
=& < 0 8 @ 3 &>9
A>9

```

2.4 Elementos para el desarrollo

Habiéndose ya establecido Ext como la librería de apoyo base para el desarrollo de los gadgets estáticos, a continuación se hace una breve reseña de las capacidades de la misma que han sido usadas para la programación de los gadgets.

2.4.1 Ext.Element

Como se ha dicho anteriormente, los gadgets se van a desarrollar en un HTML dividido por capas. Cada una de estas partes estará constituida por un elemento *div* que, a su vez, contendrá un identificador que posibilitará situar un gadget en cada uno de ellos. El método tradicional de acceso a estos elementos a través de JavaScript es:

```
7 63 1 #7%#B 7 B>9
```

En este caso se asignaría a la variable `myDiv` el elemento del HTML referenciado por el identificador `myDiv`. Este método obtiene correctamente el elemento indicado, pero el objeto devuelto no ofrece mucha funcionalidad. Para ello Ext ha creado el elemento `Ext.Element` que es realmente el corazón del framework.

```
8 2 37=< => ?  
7 6 8 1 =B 7 B>9  
A9
```

Ahora `myDiv` es una referencia a un `Ext.Element` que envuelve la mayoría de los métodos y propiedades DOM que se necesitan, proporcionando una interfaz conveniente, unificada y multi-navegador. Además de esto la llamada `Ext.get()` proporciona cacheo interno, por lo que múltiples llamadas a un mismo objeto se realizarán con mayor rapidez.

El objeto `Element` proporciona una serie de métodos variopintos como por ejemplo, resaltar el fondo del elemento, añadir una clase CSS o centrar el elemento en el viewport.

2.4.2 Ext.data.DataProxy

Es una clase abstracta, que permite crear especializaciones dedicadas a la obtención de objetos que actúan de contenedores de datos sin formato. Éstos pueden provenir, tanto de peticiones HTTP como de datos introducidos a mano por el usuario. Dependiendo de la localización de estos datos, existen unas clases predeterminadas que heredan de `DataProxy`. La clase `Ext.data.HttpProxy` permite realizar peticiones HTTP a urls que se encuentren en el mismo dominio que el JavaScript ejecutado, es decir, esta clase no permite Cross-Domain Access. Si se desea hacer peticiones a dominios fuera de la máquina de la cual se descargó el fichero JavaScript es necesario utilizar la clase `Ext.data.ScriptTagProxy`. A diferencia de estas dos últimas clases mencionadas, `Ext.data.MemoryProxy` recoge datos introducidos manualmente por el desarrollador, por ejemplo en un array.

2.4.3 Ext.data.Store

Esta clase encapsula una cache de objetos `Ext.data.Record` del lado del cliente, que proporcionan datos de entrada para los widgets tales como `Ext.grid.Grid`, o `Ext.form.ComboBox`.

Un `Store` utiliza la implementación de un `Ext.data.DataProxy` para acceder a los objetos, ya sean pasados directamente por el usuario u objetos devueltos por peticiones HTTP. El `Store` no tiene conocimiento del formato del objeto devuelto por el `DataProxy`, por lo que es necesario el uso de clases auxiliares para poder extraer el valor de los datos que contiene dicho objeto. Cuando un objeto es recibido por un `Store`, pasa a ser tratado por un `Ext.data.DataReader`. Éste extrae los datos y los almacena en un `Ext.data.Record`, el cual se encargará de guardar tanto la definición de estos datos, como sus valores. A partir de este momento si un usuario desea acceder a cualquier dato lo debe hacer a través de estos `Record` que se encuentran en la cache del `Store`.

Existen distintos tipos de `DataReader`, los cuales se diferencian en el tipo de formato que son capaces de tratar. Por ejemplo, el `Ext.data.XmlReader` únicamente es capaz de extraer datos de un XML, al igual que el `Ext.data.JsonReader` lo hace de un JSON y el `Ext.data.ArrayReader` lo realiza de un array que puede haber sido definido directamente por el usuario. Aquí se expone un ejemplo de cada uno ellos con el `Record` correspondiente:

```

" 8 3    72  3

2 3 <   6 8 3  2  3   =C
? / B    BD.. 1/ ,AD " *.   . E , 3 3  7  B  B
? / B .   BD.. 1 /FA" *.   . E  F3 3  7
                                " B .   B

G>9

72 3    6 0  8 3    72  3 =?
3/      "      .      .      3 5 .  3 7  H
AD 2 3 <>9 "    3 < 3 I  3 3

"    2 3    3J  3  3  7    < /

C C,B! 3 BDBK 3  BGD 1 BDB% <@  BGG

```

```

" 8 3    K 2  3

2 3 <   6 8 3  2  3   =C
? / B B D.. 1/B  BAD
? / B .   BA

G>9

72 3    6 0  8 3    K 2  3 =?
! . 7/ &  &D
/ & 0 &D
3/& 3&
AD 2 3 <>9

"    2 3    3J  3  3  K -    < 3 /

? B    B/FDB 0 B/C
? B 3B B  B/ B#  BD.   /BL 3  B AD
? B 3B B  B/ B#  BD .   / B)    B  AG

A

```

```

" 8 3 M 2 3

2 3 < 6 8 3 2 3 =C
? / B B D .. 1/B BAD " $ . . 3 3 B .. 1B
" M*$D .

? / B . BA "B . B
G>9

72 3 6 0 8 3 M 2 3 =?
2 3 / & 3 &D 3 1 3 M*$ =. >
3 / &< &D " 5 . 7 < E
" 5 3
3 / & 3& " 3 < 3 3 3 1 =. >
AD 2 3 <>9

" 2 3 3J 3 3 M*$ < 3 /

N8 N4
3 4
3 4 F' 3 4
< 4
34, '34
4! 3 ' 4
. 4K 3 ' . 4
'< 4
< 4
34F '34
4 1 ' 4
. 4% < @ ' . 4
'< 4
'3 4

```

Después de mostrar cómo se crearía un Reader, se va a exponer la creación de un Store que extraiga información, en este caso, de una petición HTTP:

```

3 6 0 8 3 =?
" H ) !. < E
. 87/ 0 8 3 ) ! 87=? / B B/D

"$ B B3 3 < E 3 . 3 2 3
3 / 2 3 A>9

```

2.4.4 Ext.grid.Grid

Esta clase es la representación básica de una tabla o cuadrícula. Las ventajas que aporta sobre las tablas básicas de HTML son varias. Es mucho más sencillo dar un formato y una interfaz amigables para el usuario gracias al framework. Además de esto, proporciona una serie



de características que pueden ser configuradas al generar un objeto de esta clase, como por ejemplo `autoExpandColumn`, que expande la columna indicada hasta ocupar todo el hueco libre del grid.

Los datos que se muestran en un grid son extraídos por `Readers`, es decir que se pueden obtener del propio código, por ejemplo datos que el desarrollador haya introducido en un array, de un XML que ha sido devuelto de una petición a un servidor, etc. Más abajo se expone un ejemplo que, obviando la implementación del `DataStore`, crea un tipo `ColumnModel`, usado para dar formato a cada una de las columnas del grid, y configura la tabla mediante sus propios atributos.

```

7 *3      6 0 8 1 3      *3      =C
? 3 / &%3 3 / D 3 %3 8/ B 3BAD
? 3 / &- &D 0 3 / , D 3 %3 8/ B BAD
?3/&$ &D 3 / &$ &D %3 8/ B BA

G>9
7 *3      3<          6 9
1 3 6 0 8 1 3 L 3=& 3 3      3 &D?
3/ 7      D
/ 7 *3 D
8. 3      /B$ BD
( 3 02 H      /< D
*          /

A>9
" 5      J 5      . E
1 3 3 =>g' 2 3 H      7      3 3 3

```

En este ejemplo hay varios aspectos reseñables. El primero de ellos es la composición del `ColumnModel`. La configuración está compuesta por el header, que se refiere al nombre de la cabecera de cada columna en el interfaz, el `width` que es la anchura de la columna (en la última columna no se introduce este atributo ya que más adelante se le proporcionará la propiedad `autoExpandColumn`), el `dataIndex`, que es el nombre con el cual se identifica en el `Reader` esa columna y el `id`, que asocia un nombre a la columna para más tarde poder ser identificada desde fuera (como por ejemplo en el caso de asociarle la propiedad `autoExpandColumn`).

2.4.5 Ext.form.Form

Lo mismo que ocurre con los `Grids`, la clase `Ext.form.Form` es únicamente una mejora de los formularios típicos de HTML. Como ya hemos comentado en casos anteriores, gracias a `Ext`, es más sencillo realizar un formulario con interfaz agradable. Además de esto se introducen atributos de configuración del formulario que afectan, al tipo de petición que se va a realizar al hacer un submit, al `Reader` que va a tratar la respuesta de estas peticiones, etc. También ofrece facilidades tales como la validación de casillas, que se realiza de forma sencilla y que permite al desarrollador comprobar, a nivel de cliente, si una casilla cumple con ciertas restricciones (no estar vacía, ser numérica, etc).

```

1 :      6 0 8 < : =?
          1/ B 1 BD
          (3 / OP
A>9

1 : 3 3=
      0 8 < 8: 3=?
      < 3$ / B" BD
      / B BD
      0 3 /,OPD
      0# /< D
      8/ B 1 5 3B
A>D

      0 8 < 8: 3=?
      < 3$ / B! 0 3BD
      . 7. / B. 0 3BD
      / B. BD
      0 3 /,OPD
      0# /< D
      8/ B . 0 3 5 3B
A>

>9

```

A los formularios se les pueden añadir botones de manera sencilla:

```

1 : 3 3# =B$1 BD< =>?
      " E3 1 3 < 3 E
A>9

```

Y por último es necesario renderizar el formulario en un div:

```

1 : 3 =B3 B>9

```

2.4.6Ext.BasicDialog

Clase que permite añadir cuadros de diálogo flotantes, dentro de los cuales se pueden albergar formularios, grids, y cualquier objeto perteneciente a Ext. A continuación se muestra el código que genera un cuadro de diálogo con dos botones que tienen asociada la función `dlg.hide` (cierra la ventana).

```

3 1 6 0 8 #      1=&3&D
1 /F D
0 3 / D
) 1 /, D
( 3 / ,P D
3 / D
. 87 1/ D
3 0/
A=9
3 1 33Q7$      =FOB 1 3 D3 1>9 E R 3 1
3 1 33#      =B QB 1 3 D3 1>9 " E 1 @3 < E
3 1 33#      =B B 1 3 D3 1>9
3 1 0=>9      " * 1

```

2.4.7Ejemplos

Se van a exponer tres ejemplos de gadgets que se consideran básicos para la correcta implementación de una plataforma estática. Se pretende mostrar al desarrollador como realizar, de manera sencilla, la comunicación entre gadgets.

En este caso se propone un grid, un formulario y un cuadro de diálogo. Cuando un usuario pulse sobre una casilla del primer gadget se actualizarán los datos del formulario, mientras que éste contendrá un botón el cual, al ser pulsado, hará aparecer un cuadro de diálogo que capturaré los datos del formulario.

El HTML elegido para contener estos gadgets es el siguiente:

```

!      !"#$$% &"( " ) *$ + , " - && ./ '000 0 1'2' +' 3 3&4
4
34
. 5 6&      7. &      6& 8 ' 9      6" :;&4
4      ' 4
% 3      "%      3 8 / 4
. 7.6& 8 '      .&      6& 8 ,, , ' 3 . '7 '7      &4' .4
. 7.6& 8 '      .&      6& 8 ,, , ' 3 . '7 '8      7 3.      &4' .4
. 7.6& 8 '      .&      6& 8 ,, , '8      3 1 &4' .4
. 7.6& 8 '      .&      6&      &4' .4
% 3 8 7      / 4
6& 7      &7.6& 8 ' & <6& 8 ,, , ' ' '8      &4
6& 7      &7.6& 8 ' & <6& 8 ,, , ' '7      &4
6& 7      &7.6& 8 ' & <6&      &4
' 34
374
4
4
34
3      6&1 31 &4
3      6 & S &4 F4L 3' F4' 3 4
3      6&      &36&1 3&4
'3 4
'34
34
3      6&1 31 &4
3      6 & S &4 F4: ' F4'3 4
3      6& & 36&< &4
'3 4
'34
' 4
3 36&      1&4
' 4
' 374
' 4

```

Se ha creado una tabla con una única fila y dos columnas. En cada una de ellas se sitúa un div del tipo "gadget", el cual está definido un fichero .css llamado "Tutorial.css". Dentro estas capas se encuentran otros dos divs, que diferencian el título del gadget y el contenido. Ambas partes están también definidas el .css.

En primer lugar se va a definir la tabla que va a contener los datos iniciales. Para simplificar las cosas se ha representado la información en un array, que podría ser sustituido, por ejemplo, por un XML recibido de una petición a una url.

```

7      6 CCB 1BDB% <   BDCBDB 3   B GDCB 3 BDB! 5 BGG9

3 6 0 8 3      =?
      . 87/ 0 8 3 *      7! 87= 7 >D
      3 / 0 8 3      72 3 =?ADC
      ? / B- BAD
      ? / B! < BA

      G>

A>9

6 0 8 1 3      * 3      =C
      ? 3 / &- &D 0 3 / , D3 % 3 8 / B- BAD
      ? 3 / &! < &D 3 / &! < &D 0 3 / , D3 % 3 8 / B! < BA

G>9

3 <      6 9

1 3 6 0 8 1 3 L 3=B1 3B?
      3 / 3 D
      / D
      8 . 3 / B! < B

A>9

1 3 =B 0 BD<      =1D 0D >?
      " E3 1      <

A>9

1 3 3 =>9
3 3=>9

```

Para poder actualizar los datos en el formulario es necesario definir la acción que se va a ejecutar cuando se produzca el evento deseado, en este caso la elección de una fila. Esto se consigue creando un manejador para el evento rowclick, siendo el método on el que proporciona la API de Ext para esta función. Antes de explicar el código de esta función se va a observar el del formulario:

```

< 6 0 8 < : =?
          1 / B 1 BD
          (3 / OP
A>9

< < 3 =
? 1 3 / B: BAD
0 8 < 8 : 3=?
< 3$ / B- BD
  / B BD
  3 7/ D
0 3 /,P
A>D

0 8 < 8 : 3=?
< 3$ / B! < BD
  / B. < BD
  3 7/ D
0 3 /,P
A>

>9

< 33# = &# &D =>?
      " E3 1 5 E
A>9

< 3 =B< B>9

```

El formulario está compuesto por dos campos de texto, "Nombre" y "Profesión", y por un botón que hará aparecer un cuadro de diálogo. El código de este se expondrá más adelante. Para hacer referencia al formulario y poder actualizar sus datos dependiendo de la fila seleccionada en el grid es necesario insertar el siguiente código en el interior del manejador de evento creado.

```

< T =?
      / 1 1      => 1 = 0> 1 =B- B>D
      . < / 1 1      => 1 = 0> 1 =B! < B>
A>

```

Donde g hace referencia al grid sobre el cual se ha capturado el evento y row al número de fila seleccionada. Esta función únicamente extrae los valores de la tabla y los coloca en los campos del formulario "nombre" y "profesión".

El último punto es el cuadro de diálogo, en este caso únicamente mostrará una frase en la cual estarán incluidos los valores de las casillas del formulario.

```

8 6 0 8 < 8: 3=?
0 3 / D
3 7/ D
/ B BU< < 3: 3=B B> 1T =>UB 7 3 BU
< < 3: 3=B. < B>1 T =>

A>9

3 1 6 0 8 # 1=B 1 B0
1 /FP D
0 3 / P D
) 1 /FP D
( 3 / P D
3 / D
3 0/

A>9

3 1 =B# 1B>9
8 3 =3 1 37>9
3 1 0=>9

```

Este código debe ejecutarse cuando se pulse el botón del formulario, con lo que el código debe situarse en el interior de la función que ejecuta éste.

3 DESARROLLO DE PLATAFORMA DINÁMICA

3.1 Criterios generales

El proyecto EzWeb se centra en las tecnologías clave a emplear en el desarrollo de la capa de acceso web (front-end layer) a los servicios sobre Arquitecturas Orientadas a Servicios (SOA - Service Oriented Architecture) de nueva generación que permitan dar soporte a los siguientes aspectos:

- Los usuarios finales deben contar con la máxima autonomía y capacidad de personalización en relación con la configuración de su entorno operativo, como resultado de localizar, elegir, personalizar y combinar de manera flexible y dinámica (siguiendo un modelo de autoservicio) recursos disponibles en la red.
- Los usuarios finales deben contar con la capacidad de crear y compartir conocimiento, que se materialice en la capacidad de construir nuevos recursos y publicarlos en la red, así como intercambiar experiencias con otros, aprendiendo juntos y acelerando de este modo tanto la incorporación constante de innovaciones como la mejora de la productividad.
- La interacción debe adaptarse y ser relevante al contexto, dando al término "contexto" el significado más amplio posible, de manera que comprenda elementos tales como el contexto del usuario (conocimiento, perfil, preferencias, idiomas, información sobre las redes sociales a las que el usuario pertenece, etc.) o el contexto de utilización (características estáticas y dinámicas del dispositivo usado para el acceso, localización geográfica y de tiempo, la conexión de banda ancha, etc.); y teniendo en cuenta tanto la variabilidad del contexto como la movilidad de los usuarios.

EzWeb plantea su evolución dentro de las siguientes líneas de actividad:

- Gestión avanzada del entorno operativo
- Catálogo de Recursos
- Marketplace de Recursos
- Técnicas de adquisición y explotación del conocimiento
- Entorno de Desarrollo Avanzado

La diferencia principal que plantea la plataforma dinámica con respecto a la estática es que los gadgets implementados pueden portarse a otras plataformas

3.2 Framework

EzWeb, como proyecto libre, aplica su propia filosofía de desarrollo, apoyándose en librerías JavaScript libres para su funcionamiento interno, tales como las ya mencionadas Prototype y Script.aculo.us, por lo que las funciones y envolturas habituales de estas librerías están disponibles por defecto para cualquier gadget. No obstante, implementa un conjunto propio de funciones que permite a los gadgets comunicarse entre ellos y la tarea habitual de almacenar parámetros de configuración.

3.2.1 EzWeb API

Además de las librerías de apoyo comentadas y sus respectivas APIs (Prototype⁸ y Script.aculo.us⁹), EzWeb cuenta con sus propios métodos. Si bien en la actualidad simplemente es posible el uso y manipulación de variables, ya sean de estado, preferencias o conexionado, se prevé que en un futuro crezca ofreciendo nuevos métodos útiles a los desarrolladores según crezcan las necesidades de los mismos.

⁸<http://www.prototypejs.org/api>

⁹<http://wiki.script.aculo.us/scriptaculous/>

Todas las funciones se invocan como métodos del objeto EzWebAPI, que se obtiene al incluir la librería EzWebAPI.js. Actualmente se cuenta con las siguientes funciones.

- EzWebAPI.getId(). Obtiene el identificador de la librería para el usuario.
- EzWebAPI.createRWGadgetVariable(name). Crea una variable persistente de lectura y escritura de nombre *name*. Esta función crea un objeto con los siguientes métodos.
 - get(). Obtiene el valor actual de la variable persistente.
 - set(value). Establece y persiste la variable con el valor indicado en *value*.
- EzWebAPI.createRGadgetVariable(name, set_handler). Crea una variable persistente de lectura, esto es, una variable que puede usarse en el código pero a la que no se le puede establecer el valor programáticamente, ya que suelen representar preferencias o variables de wiring. Recibe dos parámetros, *name*, que especifica el nombre de la variable que se desea leer, y *set_handler*, que indica la función a invocar cuando el valor de la variable es modificado, recibiendo como parámetro el nuevo valor. El objeto creado por esta función sólo tiene un método.
 - get(). Obtiene el valor actual de la variable persistente.
- EzWebAPI.send_get(url, context, successHandler, errorHandler).
- EzWebAPI.send_post(url, parameters, context, successHandler, errorHandler).

3.3 Implementación de gadgets

A continuación se tratarán los aspectos fundamentales de la plataforma, describiendo los elementos básicos de los que consta un gadget EzWeb. Resumidamente, un gadget en la plataforma EzWeb consta de un documento XHTML con el contenido del mismo y otro fichero, denominado *template* que mantiene información de diversa índole sobre el gadget y que enlaza al contenido XHTML del mismo.

3.3.1 Template

Una de las claves del éxito de una plataforma de desarrollo radica en la forma en que se divulga su modo de uso. La plantilla estándar de representación de recursos es la pieza más importante en el puzzle que el programador ha de componer para incorporar un gadget u otro recurso a la plataforma EzWeb.

En ese sentido, la elección y definición de la plantilla ha sido uno de los trabajos maestros de la implementación de la plataforma. Sin embargo, sin una forma completa y unívoca de expresar lo que una plantilla puede ser o no, la usabilidad del proyecto estaría en grave riesgo. Este es el propósito de esta sección: describir de forma coherente y formal la plantilla estándar de descripción de recursos.

La plantilla EzWeb responde a unas normas sintácticas firmes. Hay que recordar que es un documento que va a ser procesado automáticamente y que la plataforma va a utilizar para conocer las diferentes propiedades que ha de tener en cuenta para mostrar un gadget dado al usuario que lo utiliza. Para poder llevar a cabo esta tarea, la plantilla debe estar escrita en un lenguaje formal. En ese sentido definiremos un nuevo conjunto de normas sobre XML para que la plataforma EzWeb pueda interpretar de forma correcta la plantilla de cualquier gadget que respete dichas normas. Estas normas podrían expresarse mediante varias técnicas, pero es necesario alguna que nos permita establecer una descripción formal. Siendo este el motivo por el que se ha optado por un XMLSchema.

N8 6&, & 3 16 & <;&N4

. \$ 6& ./' ' < . 1 'F O' . &4

* 1 3 < 131 . . 4

12 . 4

T 3 4* < ' T 3 4

- 4\$ H 3 '- 4

T 4, 'T 4

4 1 ' 4

* 4 1 V. 1 < . '* 4

. 4 131 . . 3 3 3 3 ' . 4

% 1 "2%4 ./' ' . < . ' H0 '1 31 ' 1 < ' % 1 "2%4

("2%4 ./' ' . < . / O '\$ H 3 '("2%4

' 12 . 4

H(L 3 1 1 4

! < ! < 4

! < 6& 33 & 7.6& 8 &3 . 6& 33 & 6& 3 3 & '4

'! < ! < 4

H(L 3 1 ! 4

! < ! . 4

'! < ! . 4

H(L 3 1 (1 4

! < (14

(6& 33 & 7.6& 8 & 6& 33 & < 3 3 6 & 33 & 0 16& & '4

'! < (14

! < \$ 4

M) *\$ < 6& ./' ' . < . ' H0 '1 31 ' H 3 & '4

'! < \$ 4

! < 2 3 1 0 3 6&F& 1 6&F&4

' . 4



Como se puede ver en el ejemplo adjunto, la plantilla estándar consta de varios apartados. Aunque toda la información que necesita la plataforma en relación a un gadget va concentrada en una sola plantilla, no es posible incluir todos los datos de forma desordenada y desestructurada. Para facilitar tanto la lectura como la interpretación de la plantilla, los datos están agrupados según la familia semántica a la que pertenezcan, de esta forma, no cabe esperar un dato relativo a cómo se indexará el gadget o recurso en el catálogo de gadgets de la plataforma junto a otro sobre qué eventos disparará el gadget sobre la plataforma.

La clasificación de los datos se realiza mediante los siguientes grupos:

- Descripción de recursos para el catálogo (Catalog.*)
 - Datos relativos a cómo debe ser indexado el gadget (Catalog.ResourceDescription)
- Especificación de datos sobre la plataforma EzWeb (Platform.*)
 - Preferencias para la plataforma alterables por el usuario (Platform.Preferences)
 - Estado del gadget o elementos persistentes del mismo (Platform.StateProperties)
 - Elementos que participan en la interconexión del gadget con otros gadgets de la plataforma (Platform.Wiring)
 - Recursos externos que están directa o indirectamente relacionados con el gadget (Platform.Link)
 - Capa de presentación o datos relacionados con la representación del gadget sobre la plataforma (Platform.Rendering)

Cada grupo tiene capacidad para un conjunto de variables que especifican todos los datos necesarios relativos a cada grupo.

```

!      ) *$ !"#$$% &"( "      ) *$ + , " -& & ./"000 0      1'2' + '      3 3&4

4
34
. 5 6&      7. &      6& 8 ' 9      6      ;;PW ,&'4
. 1 16&      . & 6&'H0      ' 'H(      !%'H(      !% &4' .4
4T 3 3      3 5 . '      4
.4
3 % 3 6 H( !      %      2L 3 1 T      =&3      %&DS3 )      3 >9
3      6 H(      !%      2L 3 1 T      =& 3      &DS      )      3 >9
33      6 H( !%      2(L      3 1 T      =& 3 3      &>9
0 1      6 & Q&9

'X . 7      3 <      3 X '
3      6 CG9

3      6      C& $ *      $ 2 - F&D&      ,, ,, ,, ,, , &D&:      3 H      ! HD
2 3 1 & D& O+, F&D&      ,D& &      & D&*M:D&      $D&: !&D&      * 7 FPD F;WF,      &G9
3      C& 3      ,&G& 3      9
3      6      C& -* $ #-      ;&D&, , FFFF&D& 2 3 1 H      :      3      HD
&D& YO,+&D&      +&D&      &D& $ &D&$      &D&!&D&      3      +DF;,      3      &G9
3      C& 3      F&G& 3      9
3      6      C& -* $ L-      ,F&D&, ,, ,, , &D&:      3 H      ! HD
2 3 1 & D& MF, Y; PF&D&      Y&D&      &D& $ &D&$      &D&!&D&      3      *      3      F+DF;WP L < &G9
3      C& 3      &G& 3      9
3      6      C& 2 $' ) , &D      &, , , FFFF&D& 2 3 1 H      :      @      3      HD
&D& YO,F&D&F&D&      &D& $ &D&$      &D&!&D&      3      FP DF; + O* 3 3&G9
3      C& 3      +&G& 3      9

```

3 6 C& \$* 2Z Y&D& ,, ,, ,, ,, &D: 3H ! HD
 23 1 & D& PP ,F&D& F&D& & D& \$&D& \$&D&! &D& 3 FDF;F \$ 2H 3 *3 3&G9
 3 C& 3 P&G 3 9
 3 6 C& -* \$ #- ;&D& ,, FFFF&D& 2 3 1 H : 3 HD
 &D& YO,+&D& +&D& &D& \$ &D&\$ &D&:&D& 3 +DF; 3 &G9
 3 C& 3 Y&G 3 9
 3 6 C& -* \$ O&D&, ,, ,, &D&: 3H ! HD
 23 1 & D& Y , F&D& & D& &D& \$ &D&\$ &D&:&D& 2 3 *3 ,FDF;W, * &G9
 3 C& 3 O&G 3 9
 3 6 C& -* 2% 2 ,&D& ,, FFFF&D& 2 1 H : 3 HD
 &D& PPY&D& , &D& &D& \$ &D& \$ &D&:&D& 3 2 2 +YDF; *3 3&G9
 3 C& 3 ;&G 3 9
 3 6 C& 2ML : &D&, ,F FFF&D& 2 3 1 H : 3 HD
 &D& Y &D&,&D& &D& \$ &D& L&D&M&D& T 3 FPF;WF L < &G
 3 C& 3 W&G 3 9
 3 6 C& 2 \$ +&D&, ,, ,, &D&: 3H ! HD
 23 1 & D& VY YO &D& ,&D& &D& \$ &D& \$ &D&:&D& 3 ! OD F;;F 3 &G9
 3 C& 3 , &G 3 9

 < S) 3 = >
 ?
 0 1 6 9
 A

 < S3) 3 =3 >
 ?
 <=3 6 [[3 6 3 < 3>
 ?

 6&&9
 U6& . 36B B7 6\< < 7/ 9 < H/ FF.89< 0 1 / 39 1 / P.89 1 /
 P.89 /] OOG 4 3 5 . '4& 9
 U6& 7 6\< < 7/ 9 < H/ ,F.89 .33 1/ ,.89\&4&9
 U6& 4 16B <B4! . 3 3 3 5 . ' 4 16B <B4T ' 4' 4&9
 U6& 4 34 3 1 3 5 ./ ' 34 34 &U3 C3 G C G&' 34' 4&9
 U6& 4 34 3 / ' 34 34&U0 1 U&' 34' 4&9
 U6& 4 34 3 1 3 2 . / ' 34 34 &U3 C3 G C, GU&34' 4&9
 U6& 4 34- 3 2 . / ' 34 34 &U3 C3 G C F&U 34' 4&9
 U6& 4 34% / ' 34 34&U3 C3 GC GU&' 34' 4&9
 U6& 4 34- / ' 34 34&U3 C3 GC+GU&34' 4&9
 U6& 4 34 / ' 34 34&U3 C 3 G C U&' 34' 4&9
 U6& 4 34 . / ' 34 34&U3 C3 GCY GU&34' 4&9
 U6& 4 34 . < 3 / ' 34 34&U3 C 3 G O G U&34' 4&9
 U6& 4 34* 3 / ' 34 34&U3 C 3 G C U&' 34' 4&9
 U6& 4 34 / ' 34 34&U3 C 3 G W GU&34' 4' 4&9

 3 1 #7%=& <>)*\$6 9
 33 =3 C 3 G C Y 9

 A
 A
 ' . 4
 ' 34

```
374
3 36B < B4 4 . 3 5 . ' 4 '3 4
' 374
' 4
```

En el ejemplo de código fuente de gadget anterior se muestra la libertad de la que goza el programador, ya sea utilizando XHTML como el código JavaScript que desee.

3.3.1.1 Metadatos para el catálogo

Como ya se ha apuntado someramente, la plantilla estándar de descripción de recursos alberga una sección dedicada exclusivamente a la forma en que el gadget o el recurso será indexado en el catálogo de recursos.

En esta sección se incluirán los valores mediante los cuales el catálogo organizará los gadgets. Es de suponer, que de esta información depende el que un usuario recupere o no este gadget, mientras busca en el catálogo de recursos.

Dicha sección recibe el nombre de "Catalog.ResourceDescription" y contiene varios apartados que se enumeran e introducen a continuación:

- **Vendor.** Este campo contiene el nombre que identifica a la persona o institución que libera el gadget. Clasifica de forma definitiva y casi unívoca los gadgets de una misma entidad o persona, lo que los hace inconfundibles respecto a gadgets de otros productores, por muy similares que sean los enfoques utilizados en su producción.
- **Name.** Este segundo nombre, en combinación con el del fabricante, sí que identifican de forma exclusiva cada gadget. Esta información debería ser suficientemente significativa como para permitir a un usuario reconocer la funcionalidad del gadget. Si no es así, aún puede consultar el resto de metadatos que aparecen a continuación.
- **Version.** Es posible que nuevas revisiones de un gadget rompan la compatibilidad respecto a versiones anteriores (compatibilidad hacia atrás). Si un usuario necesita mantener su gadget desactualizado por algún motivo, tendrá que prestar adecuada atención a este gadget. En él se expresará, de una forma coherente la versión del gadget liberado. Cabe esperar que en él se reconozcan la rama de producción (estable, inestable, etc.) y el número de versión en sí mismo.
- **Author.** Este metadato explicita quién ha sido el desarrollador del gadget o recurso. Además de la nota de reconocimiento que cabe esperar en toda producción intelectual, este campo permite abrir una vía más de comunicación para hacer llegar el feedback de los usuarios del gadget hasta el equipo de desarrollo.
- **Mail.** Este campo está preparado para albergar la dirección de correo electrónico del autor o de la persona de contacto. Esta información resulta de estimable ayuda si el usuario desea conocer más detalles sobre el producto o si quiere hacer llegar cualquier sugerencia respecto al recurso o gadget que pueda interesar al autor o al equipo de desarrollo. Este campo complementa al metadato Author a la hora de presentar la información de contacto del recurso o gadget.
- **Description.** Es un campo de texto libre. En él aparece una descripción del gadget o recurso, en lenguaje natural, en la que se habla de la funcionalidad del mismo y se tratan las características que el desarrollador o quien lo publica cree adecuado resaltar. Es conveniente aclarar que de la calidad de esta información, depende en gran medida la popularidad del gadget o recurso y el número de usuarios que recuperan dicho elemento mientras realizan sus búsquedas y consultas al catálogo.
- **ImageURI.** Este campo contiene una URI, es decir, sigue un formato estándar para identificar un recurso de forma unívoca en Internet. En este caso, el recurso identificado ha de ser una imagen. Esta imagen debe ser un icono representativo del gadget o recurso que se está describiendo. El catálogo utilizará esta imagen en forma de vista

previa de modo que el usuario se haga una idea general respecto al aspecto del gadget o recursos.

- WikiURI. En esta ocasión, también es necesario la inclusión de una URI en este campo. Sin embargo, el recurso enlazado ha de ser un wiki. El objetivo de este wiki es proveer de un marco de expresión a los usuarios del gadget o recurso donde quepan las evaluaciones, observaciones, opiniones personales que deriven del uso del elemento en cuestión. Gracias a este wiki, los usuarios que recuperen este elemento del catálogo tendrán un valioso conjunto de terceras impresiones respecto al funcionamiento, eficacia o efectividad del gadget o recurso.

3.3.1.2 Información para la plataforma EzWeb

La plataforma EzWeb requiere ciertos datos para que la conexión entre el gadget y la plataforma sea satisfactoria. En esta sección de la plantilla, el programador deberá especificar todos aquellos aspectos relacionados con la plataforma, bien por que de ellos dependa la forma en que la plataforma deba dibujar el gadget o bien porque el comportamiento del propio gadget u otros gadgets venga definido o influenciado por alguno de estos valores.

- Platform.Preferences. En esta sección se describen mediante una sintaxis muy concreta las variables de tipo preferencia que va a manejar el recurso o el gadget. Una variable de tipo preferencia es aquella que permite ser modificada por el usuario y que a su vez, la plataforma va a almacenar de forma persistente.
 - Preference. Define cada preferencia, siendo posible especificar ciertos aspectos de dicha preferencia mediante los atributos del elemento:
 - name. Recoge el nombre de la preferencia, es la forma en que tanto la plataforma, como el programador del gadget se referirá a la preferencia
 - type. Describe la naturaleza de la preferencia, esto es, determina si dicha variable puede contener un valor entero, una cadena de caracteres, u otro tipo de datos más complejo.
 - description. Alberga un texto descriptivo sobre la preferencia de la que estamos hablando. Este texto solamente es informativo y su único uso será la construcción automatizada de formularios de edición por parte de la plataforma EzWeb.
 - label. Contiene igualmente un texto, pero de naturaleza más corta que en el atributo anterior y servirá, de igual forma, para dibujar formularios de forma automática.
- Platform.StateProperties. Recoge las variables que, al margen de ser persistentes, sólo son modificables de forma programática por el mismo código fuente del gadget o recurso. Esto significa que, a diferencia de las anteriores variables, no son modificables por el usuario del gadget.
 - StateProperty. Define cada variable de tipo propiedad de estado y, al igual que en el anterior caso, necesita de los atributos del elemento:
 - name. Recoge el nombre de la propiedad, es la forma en que tanto la plataforma como el programador del gadget se referirán a la propiedad
 - type. Describe la naturaleza de la propiedad, es decir, explicita si dicha variable puede contener un valor entero, una cadena de caracteres, u otro tipo de datos más complejo.
- Platform.Wiring. En este apartado se registran las variables cuyo impacto va a superar las fronteras del propio gadget. Las variables definidas en esta sección serán incorporadas al mecanismo de *wiring* de la plataforma EzWeb. Esto significa que otros gadgets podrán utilizarlas para recoger o escribir datos de o sobre ellas. El módulo Wiring de la plataforma EzWeb contempla la aparición de varios tipos de variables. Para discriminar entre ellos, cada elemento podrá ser desambiguado mediante el atributo adecuado.

- Event, Slot. Ambos definen una variable para el Wiring, de manera que si se ha usado Event, la variable sea entrada o lectura para el gadget, y se usa Slot, de salida o lectura/escritura. Constan de los siguientes atributos.
 - name. Recoge el nombre de la propiedad, es la forma en que tanto la plataforma, como el programador del gadget se referirá a la propiedad.
 - type. Describe la naturaleza de la propiedad explicando si dicha variable puede contener un valor entero, una cadena de caracteres, u otro tipo de datos más complejo.
 - label. Contiene un texto breve que describe, de forma somera, el propósito de la propiedad.
 - friendcode. Es un identificador que servirá para establecer conexiones programáticas en forma de atajos añadiendo semántica no formal para la anotación. Gracias al valor de este campo, es posible establecer relaciones entre gadgets sin depender del interfaz estándar que la plataforma EzWeb proporciona para llevarlas a cabo de forma ortodoxa.
 - wiring. Ofrece la forma de especificar si una preferencia es de tipo salida (*slot* o *outs*) o de tipo entrada (*events* o *ins*). La plataforma EzWeb entiende que si una variable es de tipo salida, otros gadgets podrán recoger el valor que este gadget exporte mediante dicha variable. Si por el contrario la variable es de tipo entrada, el gadget tendrá la oportunidad de importar datos que otros gadgets hayan escrito previamente en esta variable.
- Platform.Link. Permite enlazar el contenido XHTML del gadget en sí, para lo cual cuenta con una única etiqueta que describimos a continuación.
 - XHTML. Permite indicar la fuente del código del gadget. Solo tiene un único atributo.
 - href. Indica la URI donde reside el código del gadget.
- Platform.Rendering. Especifica el tamaño que ocupará inicialmente el gadget en el entorno operativo (DragBoard). Tiene dos atributos que indican tanto el ancho como el alto medidos en celdas del DragBoard.
 - width. Expresa el ancho en columnas.
 - height. Indica el alto en celdas del entorno operativo.

3.3.2 Ejemplos

El procedimiento de desarrollo de un gadget implica, por un lado, escribir el código fuente del mismo en el lenguaje de programación deseado, sin que la elección de éste implique en ninguna manera a la plataforma o a su relación con el gadget, y que además tenga una representación en lenguaje XHTML. De esta forma se universaliza la creación de los gadgets, ya que sobre todas las tecnologías de servidor existentes actualmente en el mercado, bien sea Python, Perl, Ruby, PHP o Java Servlets, se coloca una capa última de presentación con el contenido en lenguaje de marcado XHTML.

Una vez el gadget ha sido creado y está disponible en Internet a través de una URI (tanto para el template como para el propio gadget), se ha de acceder la catálogo (pestaña "My Catalogue") e introducir la URI del template en la casilla correspondiente y pulsar en "Añadir nuevo recurso", lo que hará que el gadget escogido pase a formar parte del catálogo del usuario y pueda usarlo en su entorno operativo (DragBoard) pulsando sobre el botón "Añadir instancia" del gadget en cuestión.



Figura 1: Catálogo de gadgets

Por otra parte, cuando los gadgets añadidos tienen la capacidad de comunicarse entre sí mediante *wiring* (pestaña "My Wires"), se marcan en el Wiring las entradas y salidas que recibe y publica cada gadget, respectivamente. Ambas columnas están separadas por una tercera que define los canales, esto es, el medio por el cual se puede conectar una salida a una entrada. No es posible conectar dos gadgets sin utilizar un canal. Para ello, habrá que asignarle un nombre, pulsar en el símbolo "+" y seleccionarlo. A continuación, pasando el ratón por las entradas y las salidas, se resaltarán aquellos valores que comparten un código amigo (*friendcode*) definido por el desarrollador, de manera que se indique qué variables tiene sentido conectar por ser del mismo tipo semántico. Llegados a este punto, bastará con seleccionar aquellas entradas y salidas para un canal elegido.

En la siguiente figura puede verse la vista del wiring para aquellos gadgets cuyos templates definen variables de conexionado, en amarillo se resalta el canal al pulsar sobre él y cada *friendcode* se resalta con un color.

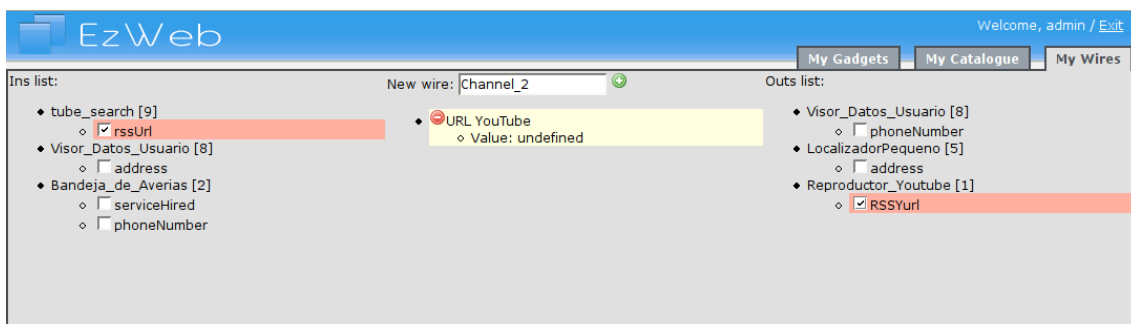


Figura 2: Canales

A continuación se van a exponer tres ejemplos de gadgets que se consideran básicos para el correcto entendimiento del proceso de desarrollo, aumentando la dificultad con cada uno de

ellos y explotando las características de la plataforma igualmente de menor a mayor de grado de necesidad/complejidad.

3.3.2.1 ¡Hola Mundo!

Como no podía ser de otra manera, el primer gadget que se mostrará será el archiconocido “¡Hola Mundo!”, de manera que el gadget producido mostrará el mensaje indicado. Para ello, lo primero que haremos será crear el código fuente del gadget, que es tan simple como un fichero HTML que simplemente escriba el texto “¡Hola Mundo!”

```

!      ) *$ !"#$$% &"( "      ) *$ + ," -&
      & ./"000 0      1'2' + ' 3 3&4

4
34
. 5 6&      7. &      6& 8 ' 9      6 < ;&'4
' 34

374
      ,4^      * 3 ' ,4
' 374

' 4

```

Con este código tan sencillo el template que lo especifica no es mucho más complejo, y de hecho, la mayor parte de los parámetros del XML quedarán en blanco, salvo la descripción para el catálogo (name, version, author, etc.), una imagen escogida para la ocasión desde Flickr, la URL donde se encuentra ubicado el código anterior y unas dimensiones.

```

N8      6&, & 3 16& <;&N
.      $      6& ./' '< . 1 'F O' . &4
*      1 3< 131 . . 4
1 2      . 4
      <Vendor>Morfeo</Vendor>
      <Name>HelloWorld</Name>
      <Version>1.0</Version>
      <Author>Javier de la Rosa</Author>
      <Mail>jrosa@yaco.es</Mail>
      <Description>El primer gadget de ejemplo.</Description>
      % 1 "2%4 . /"< . < ' W+OFWOWS O: <+< . 1 '% 1 "2%4
      ( "2%4 '( "2%4

'      1 2 . 4

! < ! < '4

! < ! . '4

! < ( 1'4

! < $ 4

```

```

<XHTML href="http://example.org/gadgets/hello_world.html"/>
! < $ 4

! < 2 3 1 width="1" height="11"4
' . 4

```

Con esto ya tenemos un primer gadget con tan sólo agregarlo al catálogo a través de su URI, que será algo como http://example.org/gadgets/hello_world.xml. Notar que para los desarrollos que se acometan, se usará la URL de ejemplo <http://example.org>; una dirección local, lógicamente, no es visible desde el exterior, por lo que en una gadget en producción tendrá que usarse un servidor con salida a Internet.

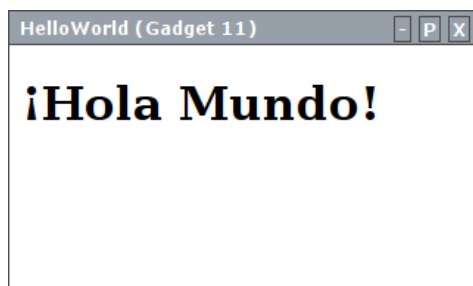


Figura 3: Gadget HelloWorld

3.3.2.2 Usando preferencias

El siguiente paso es crear un gadget que haga uso de la persistencia para almacenar datos relacionados con las preferencias del gadget o los usuarios. Como ejemplo, se creará un gadget que guarde una preferencia y luego ésta se mostrará en el código del gadget. Para conseguir esto es necesario añadir en el código del gadget la referencia a la API de EzWeb, comentada anteriormente y usar alguno de sus métodos. El código JavaScript deberá tener, por tanto, dos funciones, una que se encargue de cargar el contenido de la preferencia en el evento *onload* del código HTML del gadget, y otra que actualice el valor cuando el usuario guarde un nuevo valor de la preferencia. Además, ahora hay que expresar en el template un nuevo campo que defina la variable.

```

N8 6&, & 3 16 & <:&N4
. $ 6& ./" < . 1' F O' . &4

1 2 . 4
T 3 4* < 'T 3 4
<Name>HelloPreferences</Name>
T 4, ' T 4
4 ' 4
* 4 V7 '* 4
<Description>El primer gadget de ejemplo mejorado.</Description>
% 1 "2%4 ./" . 3 0 3 1'0 . 3 ' ' ,', ,! <
7 1' P .8 ! < 7 1 . 1'% 1 "2%4
( "2%4 '( "2%4
' 1 2 . 4

! < ! < 4

```

```

<Preference
  name="mi_preferencia"
  type="text"
  description="Esto es un texto descriptivo"
  label="Valor de la preferencia"/>
! < ! < 4

! < ! . '4

! < ( 14! < ( 14

! < $ 4
  <XHTML href="http://example.org/gadgets/preference.html" />
! < $ 4

! < 2 3 1 0 3 6&& 1 6& ,, &'4

' . 4

```

Con el template anterior, se podrá acceder a una variable de preferencia del gadget llamada *mi_preferencia*, y que será de tipo texto. El código del gadget quedaría como se muestra a continuación.

```

! ) *$ !"#$$% &"( " ) *$ + , -& & ./"000 0 1'2' + ' 3 3&4
4
34
. 5 6& 7.& 6& 8 ' 9 6 < ;&'4
% !% 3 H( 4
<script language="javascript" src="/ezweb/js/EzWebAPI/EzWebAPI.js"></script>
' 34

3 < E 3 1
3 3 . <
4
37 onload="javascript:{loadPref();}"4
,4^) ! < ' ,4
T 3 . < / <input type="text" id="pref">
' 374

<script language="javascript" type="text/javascript">
" R 3 !%
var pref = EzWebAPI.createRGadgetVariable("mi_preferencia", setPref);
" 3 * 5
var domPref = document.getElementById("pref");

" < < E 3 5 3
" 3
function setPref(value) {
  domPref.value = value;
}

" < < E 5 1 . < 1 3 1
function loadPref() {

```

```
domPref.value = pref.get();  
}  
  
</script>  
  
' 4
```

En la siguiente imagen podemos ver una muestra del gadget en funcionamiento.

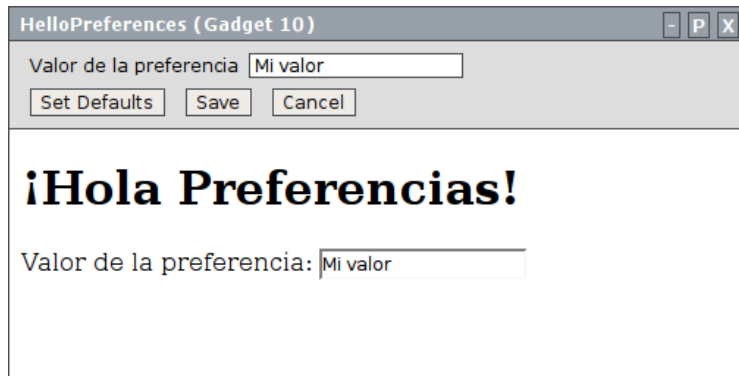


Figura 4: Gadget HelloPreferences

3.3.2.3 Usando comunicación (wiring)

Quizás, el mayor poder de la plataforma EzWeb reside en sus canales de comunicación, ya comentados con anterioridad. A continuación se mostrarán dos gadgets con sus respectivos templates: uno que publica una URL de salida al pulsar en un botón, según el texto que el usuario haya escrito en una caja de texto, y un segundo que recoge el texto y lo renderiza haciendo uso de un *iframe*. El gadget que publica será llamado URLPublisher, y el encargado de mostrar la URL indicada por el anterior, será URLBrowser.

En este caso, hemos de recurrir al apartado del template Platform.Wiring para definir la variable que hará las veces de dato de salida o entrada. Los templates de cada uno se muestran a continuación.

- URL Publisher

```

N8      6&,    &    3 16 & <;&N4
.      $      6& ./' < .    1' F O' .    &4

1  2      .    4

      T 3 4* <    'T    3 4
<Name>URLPublisher</Name>
      T 4, '    T    4
      4 '    4
      * 4    V7    '* 4
      . 4!    " 2$ . <    3 ' .    4
      % 1 "2%4    ./"000    '3    '    .1' % 1 "2%4
      ( "2%4 '(    "2%4
'      1 2      .    4

! < ! <    '4

! <    !    . '4

! < ( 14
<Event name="url" type="text" label="URL" friendcode="url_browser" />
!' < ( 14

! < $ 4
<XHTML href="http://example.org/gadgets/url_publisher.html" />
!' < $    4

! < 2    3    1 width="1" height="20"4

```

' . 4

- URL Browser

```
N8      6&,      &      3 16 & <;&N4
.      $      6& ./" < .      1' F O' .      &4

1 2      .      4
      T 3 4* <      'T      3 4
      <Name>URLBrowser</Name>
      T 4, '      T      4
      4      '      4
      * 4      \7      '* 4
      . 4*      " 2$ . <      3 ' .      4
      % 1 "2%4      ./"      '      1 ' < < 81 < '% 1 "2%4
      (      "2%4 '(      "2%4
'      1 2      .      4

! < ! <      '4

! <      ! .      '4

! < ( 14
      <Slot name="url" type="text" label="URL" friendcode="url_browser" />
!' < ( 14

! < $ 4
      <XHTML href="http://example.org/gadgets/url_browser.html" />
!' < $      4

! < 2      3      1 width="2" height="20"4

' . 4
```

Y con estas plantillas, los gadgets deben publicar el contenido de una caja de texto usando una variable de lectura/escritura y capturar las modificaciones de tal variable mediante usando una de sólo lectura.

- URL Publisher

```
!      ) *$ !"#%$ &"( "      ) *$ + , " -& & ./"000 0      1'2' +      '      3 3&4
4
34
. 5 6&      7. &      6& 8 ' 9      6 <      ;&'4
. 1 16&      . & 6&'HO      ' 'H(      !%'H(      !% &4' .4
'      34

374
      ,4"2$ !      '      ,4
      " 2$/ <input type="text" id="url_text">
```

```

        E          < E 5
    3          . 3 8
    4
    <input type="button" onclick="javascript:{setURL();}" value="Ver" />
' 374

<script language="javascript" type="text/javascript">

"          R 3 !%
var url = EzWebAPI.createRWGadgetVariable("url");
"          3 * 5 " 2$
var domURL = document.getElementById("url_text");

" <          < E 5          3
" 1l          3 . 3 8
function setURL() {
    url.set(domURL.value);
}

' .4

' 4

```

- URL Browser

```

!          ) *$ !"#%& &"( "          ) *$ + , -& & ./"000 0          1'2' + ' 3 3&4
4
34
. 5 6&          7. &          6& 8 ' 9          6 <          ;&'4
. 1 16&          . & 6&' H0          ' ' H(          !%' H(          !% &4 ' .4
' 34

374
,4"2$ # 0 ',4

<          5          @          "2$
4
URL: <div id="url_text"></div>
<iframe id="browser"
style="height: 100%; width: 100%; border: 1px #000 solid"/>
' 374

<script language="javascript" type="text/javascript">

"          R 3 !%
var url = EzWebAPI.createRGadgetVariable("url", setURL);

"          3 *
var domURL = document.getElementById("url_text");
var domBrowser = document.getElementById("browser");

" <          < E 5          3
" 7 1          .          3

```

```
function setURL(value) {  
    domURL.innerHTML = value;  
    domBrowser.src = value;  
}
```

```
' .4
```

```
' 4
```

Una vez añadidos al entorno operacional, es necesario conectarlos mediante wiring, por lo que se debe acudir a la pestaña correspondiente y conectarlos como indica la siguiente figura.

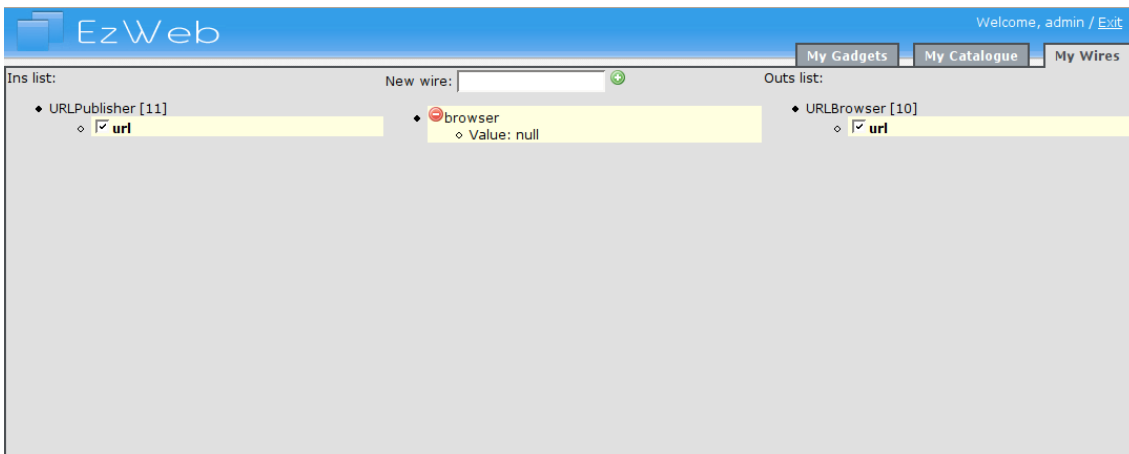


Figura 5: Conexionado de los gadgets

Una vez se haya finalizado todo, se podrá regresar al entorno operacional y visualizar los gadgets funcionando tal y como muestra la figura.



Figura 6: Gadgets conectados

4 REFERENCIAS

- Morfeo EzWeb. <http://ezweb.morfeo-project.org/>